European Control Conference 2013 (ECC'13), 16-19 July 2013, Zurich, Switzerland,

A Predictive Control Solver for Low-Precision Data Representation

Stefano Longo, Eric C. Kerrigan and George A. Constantinides

***Abstract*— We propose a method to efficiently exploit the non-standard number representation of some embedded computer architectures for the solution of constrained LQR problems. The resulting quadratic programming problem is formulated to include auxiliary decision variables as well as the inputs and states. The new formulation introduces smaller roundoff errors in the optimization solver, hence allowing one to trade off the number of bits used for data representation against speed and/or hardware resources. Interestingly, because of the data dependencies of the operations, the algorithm complexity (in terms of computation time and hardware resources) does not increase despite the larger number of decision variables.**

I. Introduction

It is common practice to optimize the performance of online optimization algorithms only at a high level of abstraction, viewing the implementation as a different, decoupled problem. This practice might no longer be reasonable due to the fact that the high degree of flexibility of new computer architectures remains largely underexploited [1]. The standard double- or single-precision floating-point representation may be unnecessarily precise for an application where such precision would be better traded in for more important aspects. For a control algorithm, these aspects could be: computational speed (it may be desirable to have fast, but less accurate results than slow, but very accurate ones), chip size (the possibility to run the same algorithm in smaller and/or less expensive platforms) and reduced power consumption.

The ability to reliably run an optimization algorithm in a low precision platform is further motivated by the fact that most microprocessors in embedded systems only offer floating-point support for single precision or half precision (10 bits) or not even provide any support for floating-point at all.

In view of the above, we propose a new formulation for the solution of the Quadratic Programming (QP) problem resulting from constrained LQR problems that allows one to exploit low precision number representations efficiently. This formulation is mathematically equivalent to the non-condensed formulation where inputs and states appear as decision variables [2]. The difference here is that we include additional decision variables in order to separate the important information stored in the system's transition matrix from quantities of non-comparable size. By doing so, the truncation due to finite precision arithmetic becomes less detrimental and the algorithm will be numerically more stable.

S. Longo is with the Department of Automotive Engineering, Cranfield University, Cranfield, MK43 0AL, UK
s.longo@cranfield.ac.uk

E. C. Kerrigan and G. A. Constantinides are with the Department of Electrical and Electronic Engineering (E. C. Kerrigan is also with the Department of Aeronautics), Imperial College London, London, SW7 2AZ, UK
{e.kerrigan, g.constantinides}@imperial.ac.uk

Delta models [3] were introduced in the 80s but they do not appear to have ever been used in industry or in the literature for the numerical solution of constrained LQR problems, possibly because such complex algorithms have historically only been implemented in platforms that support high-precision arithmetic [4]. However, the modern need for the implementation of fast *constrained* LQ solvers into inexpensive hardware (e.g. [5]) make it necessary to revive the use of discretization methods that are numerically more stable.

It will be shown that, with the proposed delta formulation, one can achieve the same closed-loop performance with fewer bits for data representation, which translates into less hardware resources needed, faster computation and less power consumption. More interestingly, it will be shown that, even with the same number of bits and despite the number of decision variables in the QP becoming larger, i) the computational cost of solving this problem still scales linearly with the horizon length, ii) the data dependencies of the algorithm are such that it can be implemented at essentially no extra hardware costs (hardware adders and multipliers are reused efficiently) and iii) the algorithm is at least as fast as its shift equivalent. In fact, our algorithm is normally faster because the better numerical stability leads to fewer interior-point iterations when an interior-point method is used, i.e. one has faster convergence to the solution.

The only price to pay for such improved performance is the usage of a small amount of extra memory for data storage (for the intermediate results) and, potentially, a slight increase in power consumption (for the increased number of operations to be performed). In a typical application these are hardly issues; the intermediate results are a small number of matrices and vectors of the size of the plant model's matrices, and the increase in power consumption for the extra additions and multiplications is negligible or nonexistent, depending on the implementation. Nevertheless, it should be noted that memory and power consumption increase with the number of bits used for data representation and the circuit clock frequency, respectively. Consequently, memory usage can be recovered by allocating fewer bits for data representation (since the same closed-loop performance can be achieved with lower precision) and power consumption can be recovered by slowing down the circuit's clock (this is possible because fewer interior-point iterations and fewer bits for the data representation reduce the computational delay).

II. Constrained LQR problem formulation in delta domain

Consider the continuous-time LTI plant model

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{1}$$

where $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$ and (A_c, B_c) is a stabilizable pair. The control input $u(t)$ is a signal created by a sample-and-hold element for a sampling period h such that, $u(t) = u(ih)$, for all $t \in [ih, ih + h)$ and $i \in \mathbb{N}_0$ is the sampling instant. The model in (1), for the discrete sampling instants i, can be written as

$$x(ih+h) = \underbrace{e^{A_c h}}_{\triangleq A_s} x(ih) + \underbrace{\left[\int_0^h e^{A_c(h-\tau)} B_c d\tau\right]}_{\triangleq B_s} u(ih), \quad (2)$$

where the matrix exponential $e^{A_c h}$ is defined by the matrix series

$$e^{A_c h} \triangleq I + A_c h + \frac{(A_c h)^2}{2!} + \frac{(A_c h)^3}{3!} + \dots. \quad (3)$$

If the product $A_c h$ in (3) results in a matrix with entries much smaller than one, the transition matrix A_s in (2) will be a matrix where the elements on the diagonal are the summation of 1 with a much smaller number. These coefficients have to be stored in a computer with finite word length. In finite arithmetic, the coefficients will be truncated, hence some of the information contained in $A_c h + \frac{(A_c h)^2}{2!} + \frac{(A_c h)^3}{3!} + \dots$ — which is where the plant dynamics are represented — will be lost. This numerical problem can be ameliorated by rewriting (2) (by substituting (3) into it) as

$$x(ih+h) = x(ih) + h \underbrace{\left[A_c + \frac{A_c^2 h}{2!} + \frac{A_c^3 h^2}{3!} + \dots\right]}_{\triangleq A_\delta} x(ih) + h \underbrace{\left[B_c + \frac{A_c h B_c}{2!} + \frac{A_c^2 h^2 B_c}{3!} + \dots\right]}_{\triangleq B_\delta} u(ih), \quad (4)$$

which is mathematically equivalent but, separating the identity matrix from the summation tail — defined as A_δ in (4) — has the effect of reducing the numerical errors in a floating-point representation.

By adopting the more convenient notation $x(ih) \triangleq x_i$ (and similarly for other vectors) and introducing a new vector $\delta_i \in \mathbb{R}^{n_x}$, we can rewrite (4) as

$$\delta_i = A_\delta x_i + B_\delta u_i \quad (5a)$$

$$x_{i+1} = x_i + h\delta_i, \quad (5b)$$

which is in fact the *delta* (or incremental) form of (1) introduced in [3], [6]–[8] — an alternative to the commonly used *shift* form of (2).

Associated with the system in (5), consider the discrete-time finite-horizon LQ problem defined by the cost function

$$V(\cdot) \triangleq x_N' Q_f x_N + \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}' \begin{bmatrix} Q & M \\ M' & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \quad (6)$$

where N is the number of samples for the prediction horizon, $\begin{bmatrix} Q & M \\ M' & R \end{bmatrix} \geq 0$, $Q = Q' \geq 0$, $R = R' > 0$, and $Q_f = Q_f' > 0$. These matrices are given to define a controller for an ideal closed-loop performance of the discrete-time system. Alternatively, they can be computed via discretization of a continuous-time LQ problem (see [9], [10, pp. 411–412]).

For the constrained LQR problem we assume full state feedback and we suppose that constraints exist on the input moves and on the states. They are represented by $(u_i, x_i) \in \mathbb{X} \subseteq \mathbb{R}^{n_u} \times \mathbb{R}^{n_x}$ where, in order to solve the QP problem, we assume that constraints lie in a polyhedral set, i.e. we can write them as $Jx_k + Eu_k \leq d$. Given a measurement or estimate of the current state $\hat{x} \triangleq x_0$ and an input sequence $(u_0, u_1, \dots, u_{N-1})$, let x_k, $k \in \mathbb{N}_0$, be the predicted state after k samples. The optimal control problem to solve is

$$\min_\theta \quad \frac{1}{2} V(\theta) \quad (7a)$$

$$\text{s.t.} \quad x_0 = \hat{x} \quad (7b)$$

$$\delta_k = A_\delta x_k + B_\delta u_k \quad (7c)$$

$$x_{k+1} = x_k + h\delta_k \quad (7d)$$

$$Jx_k + Eu_k \leq d \quad (7e)$$

$$\forall k \in \{0, 1, \dots, N-1\}, \quad (7f)$$

where θ is defined below. Problem (7) can be written as a QP problem of the form

$$\min_\theta \frac{1}{2}\, \theta' H \theta + \theta' c, \quad \text{s.t.:} \quad F\theta = f,\ G\theta \leq g. \quad (8)$$

Although we generally only wish to find the sequence of control inputs $(u_0, u_1, \dots, u_{N-1})$, it has been shown [2], [11], [12] that including the states $(x_1, x_2, \dots, x_N)$ in θ results in the QP matrices having a particular structure, which is desirable for fast online optimization when N is large. In our formulation, we not only include the states as decision variables, but also the auxiliary variables $(\delta_0, \delta_1, \dots, \delta_{N-1})$. This is done in order to preserve the information contained in A_δ and B_δ. Thus, we let

$$\theta \triangleq \begin{bmatrix} u_0' & \delta_0' & x_1' & u_1' & \delta_1' & x_2' & \dots & x_N' \end{bmatrix}',$$

where $\theta \in \mathbb{R}^{N(n_u+2n_x)}$. The associated matrices for the QP in (8) can be constructed accordingly. Input rate, endpoint and soft constraints are not included for simplicity, however, they can be added as in [2]. An added benefit of this formulation is that state rate constraints can be easily included.

III. Solution by an interior-point method with block elimination

Since the QP in (8) has to be solved in an embedded platform with limited computational power, it is reasonable to compute its solution using an interior-point method so that the algorithm works with matrices with a fixed structure (unlike active-set methods). In particular, we will consider a well-established primal-dual interior-point method with infeasible start [2], [12]. We skip all the details and point out that the most computationally expensive part of the algorithm is the solution of the linear systems for the calculation of the search direction. One can now exploit the banded structure of the Hessian and the constraints matrices and construct an equivalent system

$$\mathcal{A}\xi = \beta, \quad (9)$$

where

$$\mathcal{A} \triangleq \begin{bmatrix} R_0 & B' & & & & & & & & \\ B & & -I & & & & & & & \\ & -I & & hI & & & & & & \\ & & hI & & -I & & & & & \\ & & & -I & Q_1 & M_1 & A' & & I & \\ & & & & M_1' & R_1 & B' & & & \\ & & & & A & B & & -I & & \\ & & & & & & -I & & hI & \\ & & & & I & & & hI & & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & Q_N \end{bmatrix},$$

$$\xi \triangleq \begin{bmatrix} \Delta u_0' & \Delta\gamma_0' & \Delta\delta_0' & \Delta\lambda_1' & \Delta x_1' & \Delta u_1' & \Delta\gamma_1' & \Delta\delta_1' & \Delta\lambda_2' & \cdots & \Delta x_N' \end{bmatrix}',$$

$$\beta \triangleq \begin{bmatrix} (r_0^u)' & (r_0^\gamma)' & (r_0^\delta)' & (r_1^\lambda)' & (r_1^x)' & (r_1^u)' & (r_1^\gamma)' & (r_1^\delta)' & (r_2^\lambda)' & \cdots & (r_N^x)' \end{bmatrix}'.$$

If we let $P_k \triangleq Q_k$ and $p_k \triangleq r_k^x$, we notice that the system in (9), for $k = 2, 3, \ldots, N$ (i.e. excluding the top four block rows), is composed by the blocks

$$\begin{bmatrix} -I & Q_{k-1} & M_{k-1} & A_\delta' & & I & & \\ & M_{k-1}' & R_{k-1} & B_\delta' & & & & \\ & A_\delta & B_\delta & & -I & & & \\ & & & -I & & hI & & \\ & I & & & hI & & -I & \\ & & & & & -I & P_k & \end{bmatrix} \begin{bmatrix} \Delta\lambda_{k-1} \\ \Delta x_{k-1} \\ \Delta u_{k-1} \\ \Delta\gamma_{k-1} \\ \Delta\delta_{k-1} \\ \Delta\lambda_k \\ \Delta x_k \end{bmatrix}$$
$$= \begin{bmatrix} (r_{k-1}^x)' & (r_{k-1}^u)' & (r_{k-1}^\gamma)' & (r_{k-1}^\delta)' & (r_k^\lambda)' & (p_k)' \end{bmatrix}'. \quad (10)$$

By doing a series of block eliminations in (10), we get

$$\begin{aligned} \Delta u_{k-1} &= (R_{k-1} + h^2 B_\delta' P_k B_\delta)^{-1}((r_{k-1}^u + h^2 B_\delta' P_k r_{k-1}^\gamma \\ &\quad + B_\delta' r_{k-1}^\delta + hB_\delta' P_k r_k^\lambda + hB_\delta' p_k) \\ &\quad - (M_{k-1}' + h^2 B_\delta' P_k A_\delta + hB_\delta' P_k)\Delta x_{k-1}), \quad (11a) \\ \Delta\gamma_{k-1} &= h^2 P_k(B_\delta \Delta u_{k-1} + A_\delta \Delta x_{k-1} - r_{k-1}^\gamma) \\ &\quad - hP_k(\Delta x_{k-1} - r_k^\lambda) - hp_k - r_{k-1}^\delta, \quad (11b) \\ \Delta\delta_{k-1} &= A_\delta \Delta x_{k-1} + B_\delta \Delta u_{k-1} - r_{k-1}^\gamma, \quad (11c) \\ \Delta\lambda_k &= p_{k-1} - P_{k-1}\Delta x_{k-1}, \quad (11d) \\ \Delta x_k &= \Delta x_{k-1} + h\Delta\delta_{k-1} - r_k^\lambda, \quad (11e) \end{aligned}$$

where

$$\begin{aligned} P_{k-1} &\triangleq Q_{k-1} + P_k + h^2 A_\delta' P_k A_\delta + hA_\delta' P_k + hP_k A_\delta \\ &\quad - (M_{k-1} + h^2 A_\delta' P_k B_\delta + hP_k B_\delta)(R_{k-1} \\ &\quad + h^2 B_\delta' P_k B_\delta)^{-1}(M_{k-1}' + h^2 B_\delta' P_k A_\delta + hB_\delta' P_k), \quad (12a) \\ p_{k-1} &\triangleq r_{k-1}^x + P_k r_k^\lambda + p_k + h^2 A_\delta' P_k r_{k-1}^\gamma + A_\delta' r_{k-1}^\delta, \\ &\quad + hA_\delta' P_k r_k^\lambda + hA_\delta' p_k + hP_k r_{k-1}^\gamma - (M_{k-1} + \\ &\quad h^2 A_\delta' P_k B_\delta + hP_k B_\delta)(R_{k-1} + h^2 B_\delta' P_k B_\delta)^{-1} \\ &\quad (r_{k-1}^u + h^2 B_\delta' r_k^\gamma + B_\delta' r_{k-1}^\delta + hB_\delta' P_k r_k^\lambda + hB_\delta' p_k). \quad (12b) \end{aligned}$$

We now apply the block elimination to the first five block rows and columns of $\mathcal{A}$ with P_1 instead of Q_1 and p_1 instead of r_1^x, and derive the following equations:

$$\begin{aligned} \Delta u_0 &= (R_0 + h^2 B_\delta' P_1 B_\delta)^{-1}(h^2 B_\delta' P_1 r_0^\gamma + hB_\delta' P_1 r_1^\lambda \\ &\quad + hB_\delta' p_1 + B_\delta' r_0^\delta + r_0^u), \quad (13a) \\ \Delta\gamma_0 &= h^2 P_1 B_\delta \Delta u_0 - h^2 P r_0^\gamma - hP_1 r_1^\lambda - hp_1 - r_0^\delta, \quad (13b) \\ \Delta\delta_0 &= B_\delta \Delta u_0 - r_0^\gamma, \quad (13c) \\ \Delta\lambda_1 &= hP\Delta\delta_0 - P_1 r_1^\lambda - p_1, \quad (13d) \\ \Delta x_1 &= h\Delta\delta_0 - r_1^\lambda. \quad (13e) \end{aligned}$$

The algorithm to calculate the vector of search direction (the solution of (9)) is listed in Algorithm 1.

Algorithm 1 Compute search direction (i.e. solve $\mathcal{A}\xi = \beta$)

Input: $\mathcal{A}$, β **Output:** ξ

Algorithm:

1: Set $P_N \leftarrow Q_f$ and $p_N \leftarrow r_N^x$
2: Compute P_k and p_k for $k = N, N-1, \ldots, 2$ using the two backward recursions in (12)
3: Compute Δu_0, $\Delta\gamma_0$, $\Delta\delta_0$, $\Delta\lambda_1$ and Δx_1 using (13)
4: Compute, recursively, Δu_k, $\Delta\gamma_k$, $\Delta\delta_k$ for $k = 2, 3, \ldots, N-1$ and $\Delta\lambda_k$ and Δx_k for $k = 1, 2, \ldots, N$ using (11)

The equations derived in this section for solving the linear system in (9) are analogous to the ones presented in [2], which were derived for a vector of decision variables containing the sequence of future inputs and states only, using the shift form. Here, by also including the δ_k as decision variables, we have formulated an equivalent algorithm in delta domain.

While the block reduction manipulations of [2] yielded the famous Discrete Riccati Difference Equation (DRDE) for time-varying weighting matrices [2, eq. 51a], we obtained in (12a), analogously, a DRDE for time-varying weighting matrices in delta domain. The reader familiar with the work of [2] will also appreciate more subtle similarities between the recursive equations of both formulations, realizing that the two methods are entirely equivalent in an infinite precision arithmetic system. It is evident, however, that the solution in delta domain requires more operations (matrix additions and multiplications) to be performed at each iteration when compared to the equivalent solution in shift domain of [2].

IV. Algorithmic properties

Although computing the interior-point search direction requires more operations for the delta formulation, neither the computational time nor the hardware resources utilized significantly increase in a custom hardware implementation. The reason is twofold and lies in the dependencies of the operations.

Observation 4.1: Let Algorithm 1 be implemented in a parallel hardware architecture with a sufficient degree of parallelism. Then, when compared to the shift formulation of [2]:

(i) the critical path (the longest non-parallelizable sequence of operations) remains unchanged, hence, the time required to calculate the search direction (latency) is the same for both formulations

(ii) both formulations require the same number of computational hardware resources (adders, multipliers, etc.) for implementation.

Sketch of proof. The most computationally expensive operation of the algorithm is the calculation of matrices $P_1, P_2, \ldots, P_{N-1}$ and vectors $p_1, p_2, \ldots, p_{N-1}$ using the backward recursions in (12). For the sake of simplicity, we will only consider the DRDE of (12a) and point out that the same reasoning and procedures can be applied to the whole algorithm by also noticing that many intermediate results can be stored and reused.

(i) Let us compare the DRDE in (12a) with the equivalent shift-domain DRDE [2]

$$P_{k-1} = Q_{k-1} + A_s' P_k A_s + (M_{k-1} + A_s' P_k B_s) \\ (R_{k-1} + B_s P_k A_s)^{-1} (M_{k-1}' + B_s' P_k A_s) \quad (14)$$

and define $\bar{A} \triangleq hA_\delta$ and $\bar{B} \triangleq hB_\delta$ for the delta case and $\bar{A} \triangleq A_s$ and $\bar{B} \triangleq B_s$ for the shift case. The data dependencies of the calculations to compute P_{k-1} for both the delta and the shift formulation are shown as a graph in Figure 1. At a matrix level, Figure 1 is an implementation with minimal latency, signifying that all the operations are performed as soon as the data are available (exploiting the parallelism). The seven stages required to compute the solution are determined by the sequence of operations in the *critical path*, which is the path on the right of both graphs indicated by thicker arrows. The critical paths for both the delta and the shift case are the same, hence the time required by both algorithms to complete is the same (although matrices $\bar{A}$ and $\bar{B}$ are different for the two cases, what matters here is their dimensions, which do not change). The search direction is computed recursively in N steps in both formulations.

(ii) For the delta case, there are four extra matrix additions to be performed, shown in Figure 1 by the boxes with dotted edges. At each stage of the algorithm, a maximum of three matrix adders and three matrix multipliers are used simultaneously. This is true for both the delta and shift case, implying that the same number of hardware blocks are needed for both implementations (of course, in the delta case, the adders will be used more often). Furthermore, the same adder and multiplier blocks (as well as many intermediate results) can be re-utilized efficiently for the calculation of (11). ■

Observation 4.1 shows that the extra operations required for the delta case do not increase the algorithm latency nor the computational hardware resources utilized. Hence, the numerical advantages given by the delta implementation are obtained at no extra costs from the point of view of resource utilization and computational time.

It should also be noted that for multi-input plant models ($n_u > 1$) the bottleneck of the algorithm is the n_u-by-n_u matrix factorization needed in Stage 4 of Figure 1, which is performed on matrices of the same size in both formulations. If such a factorization is obtained via a Cholesky decomposition (followed by back and forward substitutions for the solution of the linear system), a number of division and square roots have to be computed. These operations are much more expensive to perform than additions and multiplications. This motivates even further the use of a reduced number of bits for data representation to accelerate the algorithm. As a consequence, although Observation 4.1 is valid for parallel computing architectures, an implementation of Algorithm 1 in a sequential architecture can also have an equal latency and resources utilization if a low number of bits is used. This will be shown via an example in Section V.

V. Illustrative examples

In this section we show, with simulation examples, that the solution from the delta formulation is indeed more numerically accurate than the equivalent shift one; therefore, the same closed-loop performance can be achieved with our formulation using fewer bits for number representation. An indication of the improvement in hardware resources utilized will be given for an available FPGA from the Xilinx Virtex-6 family [13]. Furthermore, simulations have shown that the increased numerical accuracy also improves the interior-point algorithm convergence speed, thus reducing the number of interior-point iterations required to completion.

We use as a benchmark a spring-mass system described by (1) with matrices

$$A_c \triangleq \begin{bmatrix} 0 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -2 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad B_c \triangleq \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},$$

having input constraints $||u_k||_\infty \leq 0.5$ and state constraints $||x_k||_\infty \leq 3.5$. The associated performance measure is given by the LQ cost in (6) with matrices Q, R, M and Q_f coming from the discretization, with sampling period $h = 0.01$s, of a continuous-time LQ cost with matrices $Q_c = R_c = I$, $M_c = 0$ and $Q_{f,c}$ as the solution of the continuous-time algebraic Riccati equation. The prediction horizon was selected as $N = 200$.

We first compare the variation of closed-loop performance when the QP solver is implemented using the proposed formulation (Section III) and the formulation of [2] (shift). As a performance metric we use

$$\Gamma \triangleq \sum_{k=0}^{N_{\text{sim}}-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}' \begin{bmatrix} Q & M \\ M' & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}, \quad (15)$$

where the system is left to evolve from initial conditions $x_0 = \begin{bmatrix} 0 & 0 & 3.5 & 3.5 & 0 & 0 \end{bmatrix}'$, $u_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}'$ and with $N_{\text{sim}} = 20/h$ (selected to be long enough to allow the system to reach steady-state). A cost, Γ_{52}, has been evaluated as in (15) for a shift implementation in double precision and for an implementation with reduced precision for data representation, which we call Γ_{low}. An error, called *closed-loop cost error* has been calculated as a normalized

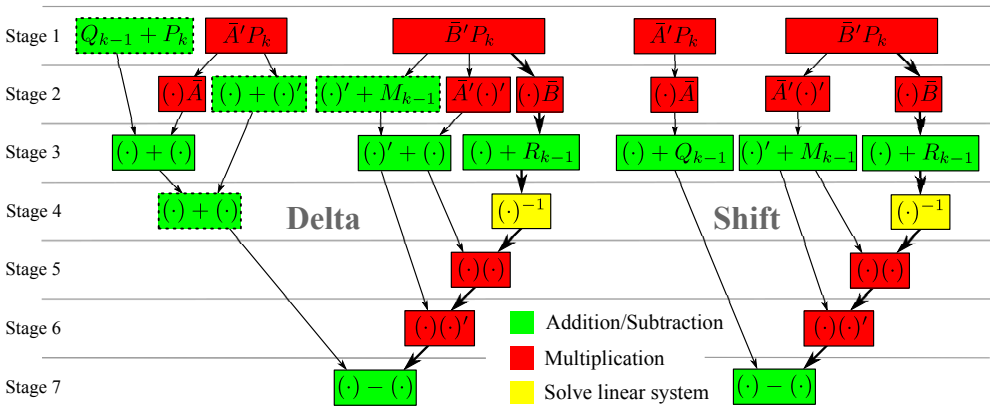

Fig. 1. Data dependencies of the delta- and shift-domain Riccati recursions. Both algorithms have the same latency because they have the same critical path (thicker arrows). At each stage, no more than three additions and three multiplications are performed simultaneously. Hence, although four extra additions are needed for the delta case (dotted boxes), the number of computational resources needed for minimal latency is the same in both cases.

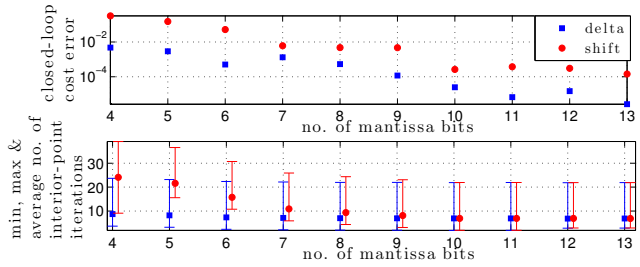

Fig. 2. Performance comparison between the delta and the shift implementation, using the Riccati recursion, for variable precision of number representation. For the same number of bits, the closed-loop cost error for the delta implementation is lower (top plot) as well as the number of interior-point iterations (bottom plot).

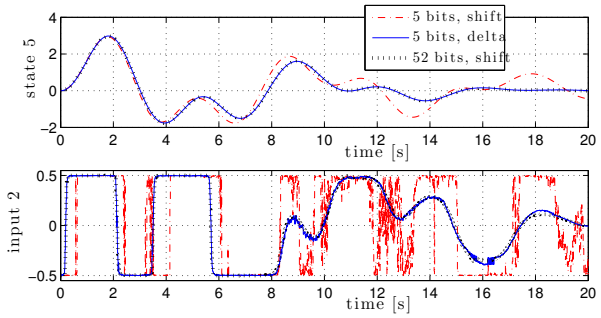

Fig. 3. Some trajectories of the closed-loop system. While the performance of a 5-bit shift implementation is very poor, a 5-bit delta implementation produces trajectories that almost perfectly overlap the ones from a 52-bit implementation.

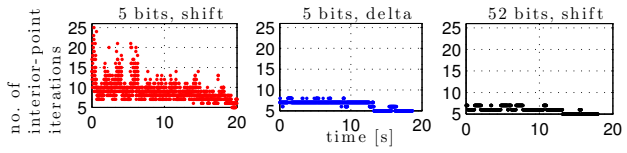

Fig. 4. Number of interior-point iterations required to completion. Because the numerical errors in the delta formulation are lower, the number of iterations required are also lower when compared to the shift implementation.

discrepancy between the double (Γ_{52}) and the reduced (Γ_{low}) precision representation, i.e.

$$\text{closed-loop cost error} \triangleq \frac{||\Gamma_{\text{low}} - \Gamma_{52}||}{||\Gamma_{52}||}.$$

The top plot of Figure 2 shows how the closed-loop cost error varies for a range of mantissa bits representation for the two Riccati recursion-based implementations. This error, for the delta implementation, is always lower than for the shift implementation, especially for a low number of bits. As the number of bits grows, the difference between the two formulations becomes negligible (hence results are shown only up to 13 bits). The bottom plot of Figure 2 shows the minimum, maximum and average number of interior-point iterations required for the duality gap to be $\mu < 10^{-5}$ (set as a tolerance to stop the interior-point iterations). The delta formulation requires on average fewer iterations to reach the predefined threshold, implying that faster convergence to the solution can be achieved.

The results from a closed-loop Riccati recursion simulation where only a 5-bit mantissa was used (and everything else left the same as before) are shown in Figure 3. While, even for such a low precision, the trajectories of the delta implementation almost completely overlap with the shift double-precision (52 bits) one, the 5-bit shift response becomes practically unacceptable. The numerical errors in the control algorithm are primarily due to errors in the evaluation of the Riccati equation, as has also been observed in [7]. Consequently, the number of interior-point iterations is also much lower for the delta formulation. This is shown in Figure 4.

Finally, we comment on the memory resources required by each implementation. Recall that, for both the Riccati recursion algorithms, the resources needed to perform the operations (additions, multiplications, etc.) and the algorithm latency are identical. The difference is the small amount of extra memory required by the delta implementation to store the increased number of intermediate results. These intermediate results are the four matrices resulting from the four extra additions (see Figure 1) and some vectors

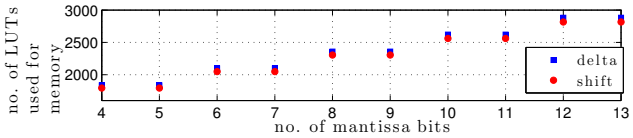

Fig. 5. Number of memory resources (LUTs) required for a given number of mantissa bits. The delta implementation always requires slightly more memory. However, it is sufficient to reduce the number of mantissa bits of at most two to bring the number of LUTs required by the delta implementation below the number of LUTs required by the shift implementation.

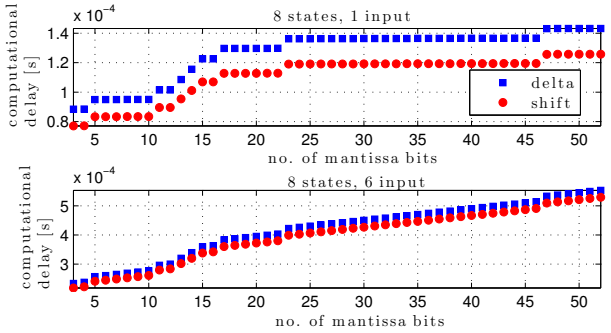

Fig. 6. Computational delay for a sequential implementation of the delta and shift algorithm (Virtex-6 FPGA, 200 MHz clock frequency). In this case, the delta implementation has a longer delay that can be recovered by reducing the number of bits.

arising from (11). We assume that the data words are stored in look-up tables (LUTs) and that 8 bits are used for the exponent part of the floating-point numbers. Figure 5 shows the number of LUTs required by both algorithms for different numbers of mantissa bits. As expected, the delta implementation requires slightly more LUTs, hence slightly more resources. However, it is sufficient to reduce the number of mantissa bits by at most two to allow the delta implementation to utilize fewer resources than the shift one. Reducing the mantissa bits affects the closed-loop performance, but the top plot of Figure 3 shows that a reduction of two bits still leaves the delta implementation with a better performance than the shift one. If, on the other hand, we assume that the algorithm is implemented on a sequential computing platform, then the delta formulation will inevitably have a bigger computational delay due to the increased number of operations. However, since the bottleneck of both algorithms is the n_u-by-n_u matrix factorization (discussed in Section IV) we will expect only a slighlty longer delay for the delta implementation for systems with more than one input. The increment on this delay is shown in Figure 6 for two spring-mass oscillators, one with one input and the other with six inputs. From the figure it is possible to see how many fewer bits a delta implementation must have in order to have the same computational delay as for a shift implementation. This is certainly more favorable for the case of six inputs, where most of the delay is due to divisions and square root operations (of which the number is the same for both implementations).

VI. Conclusions

We have shown that the superior numerical properties of a delta-domain formulation for the solution of a constrained LQR problem can be enjoyed without sacrificing the algorithm complexity in terms of its execution time or computational hardware resources required. For the Riccati recursion approach, the number of decision variables in the resulting QP problem is larger when compared to its equivalent shift formulation but, because of the particular data dependencies in the algorithm, the latency and hardware blocks required for the operations will not increase when implemented in custom hardware. The proposed design procedure give the designer the flexibility to reduce the number of bits for data representation in order to increase the computational speed and/or reduce the circuit size. This is a contribution toward the larger research goal to develop model predictive control algorithms with a guarantee that the loss of accuracy due to finite precision effects is minimized or bounded a priori.

VII. Acknowledgments

This research has been supported by the EPSRC grant number EP/G031576/1, EP/F041004/1 and the European Union Seventh Framework Programme FP7/2007-2013 'EMBOCON' under grant agreement number FP7-ICT-2009-4 248940.

References

[1] G. A. Constantinides, "Tutorial paper: Parallel architectures for model predictive control," in *Proc. European Control Conference 2009*, Budapest, HU, 2009, pp. 138–143.

[2] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *J. Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998.

[3] R. H. Middleton and G. C. Goodwin, "Improved finite word length characteristics in digital control using delta operators," *IEEE Trans. Automatic Control*, vol. AC-31, no. 11, pp. 1015–1021, 1986.

[4] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[5] A. Wills, D. Bates, A. Fleming, B. Ninness, and S. Moheimani, "Model predictive control applied to constraint handling in active noise and vibration control," *IEEE Trans. Control Systems Technology*, vol. 16, no. 1, pp. 3–12, 2008.

[6] R. H. Middleton and G. C. Goodwin, *Digital Control and Estimation: A Unified Approach*. Prentice-Hall, 1990.

[7] G. C. Goodwin, R. H. Middleton, and H. V. Poor, "High-speed digital signal processing and control," *Proc. IEEE*, vol. 80, no. 2, pp. 240–259, 1992.

[8] G. C. Goodwin, J. I. Yuz, J. C. Agüero, and M. Cea, "Sampling and sampled-data models," in *Proc. American Control Conference*, Baltimore, USA, 2010, pp. 1–20.

[9] A. H. Levis, R. S. Schlueter, and M. Athans, "On the behaviour of optimal linear sampled-data regulators," *Int. J. Control*, vol. 13, no. 2, pp. 343–361, 1971.

[10] K. Åström and B. Wittenmark, *Computer-Controlled Systems*, 3^{rd} ed. Prentice-Hall, 1997.

[11] S. J. Wright, "Interior point methods for optimal control of discrete time systems," *J. Optimization Theory and Applications*, vol. 77, no. 1, pp. 161–187, 1993.

[12] ——, "Applying new optimization algorithms to model predictive control," in *J.C. Kantor, C.E. Garcia, B. Carnahan (Eds.), 5^{th} Int. Conf. Chemical Process Control, CACHE, AIChE*, Lake Tahoe, CA, 1997, pp. 147–155.

[13] Xilinx. (2013, Apr.) www.xilinx.com. [Online]. Available: http://www.xilinx.com/

2013-07-22

A predictive control solver for low-precision data representation

Longo, Stefano

Institute of Electrical and Electronics Engineers

Longo, S., Kerrigan, E. C., Constantinides, G. A. (2013) A predictive control solver for low-precision data representation, European Control Conference 2013 (ECC'13), 16-19 July 2013, Zurich, Switzerland

http://cas.ee.ic.ac.uk/people/gac1/pubs/StefanoECC13.pdf